

Automated Transformation of Legacy Systems

Philip Newcomb and Randy A. Doblar
The Software Revolution, Inc.

The transformation of system applications code and database at automation levels exceeding 99 percent is now a viable approach to legacy information system modernization. The benefit of the approach is migrating the legacy system to a modern computing environment while preserving the repository of business knowledge and processes imbedded in the legacy system.

During the last 50 years, information processing systems have become the intellectual repositories for most business and government organizations. Today these organizations face the complex and costly problem of how best to restructure the installed base of outdated information processing resources while maintaining their legacy intellectual property. This legacy intellectual property continues to provide value as organizations are forced to innovate to survive in the fast-paced age of e-business, e-communication, e-organizations, and in the case of the military, e-warfare.

The need to modernize legacy systems is primarily driven by three factors: expansion of the system's functionality; improved maintainability of the system using modern tools and techniques; and reduction of operational costs and improved reliability by replacing obsolete hardware suites with high-speed, open-architecture systems. Alternative solutions for modernization of the system include developing a new system, system replacement with a commercial off-the-shelf (COTS) solution, or manual rewriting of the legacy applications software and databases to operate within a modern computing environment.

Developing a totally new system or replacing legacy systems by manually rewriting the system's software with the support of semi-automated tools is

extremely costly and time consuming. Replacing the system using COTS technologies, while less costly and timelier, usually requires extensive and expensive customization to provide functionality not provided by the COTS product. In addition, the Gartner Group has shown that the success rate for information system modernization projects using these traditional solutions has thus far been approximately 7 percent; it is a success rate that has not bred confidence within the information technology (IT) community.

Manual approaches have been prone to failure due to inconsistency, cost overruns, and schedule delays. COTS solutions have fallen short of expectations because of their inability to provide the customer with the functionality needed to meet its specific organizational goals. And semi-automated tool-based solutions, while relatively promising, have not provided a sufficient level of automation to overcome the drawbacks associated with manual intervention required to address untransformed code.

Looking more closely at the automated transformation approach, it becomes evident that using available tools capable of transforming 60 percent or less of a system's legacy code automatically results in extensive amounts of untransformed code that must be addressed manually. Using a one million-line information system as an

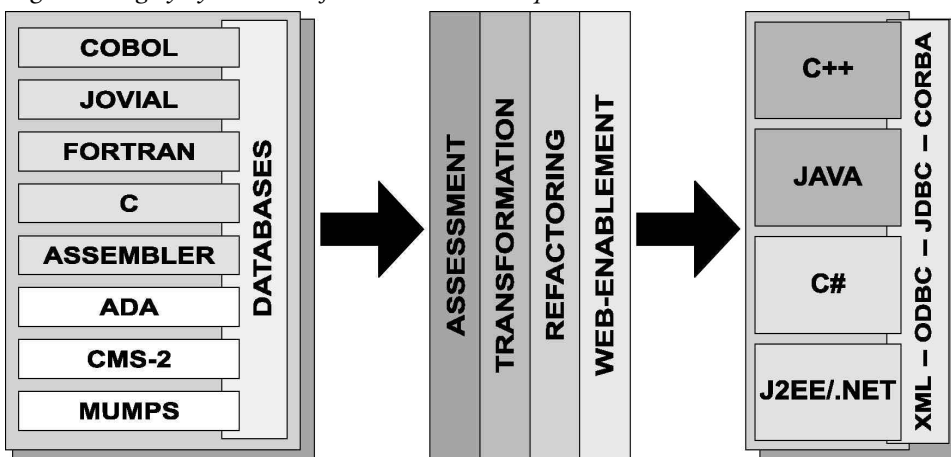
example, 400,000 lines of code would remain to be addressed manually.

The Gartner Group, an industry-recognized source of business and technology intelligence, states that on average a well-trained programmer can transform 160 lines of code per day. Continuing to use a one million-line information system and a transformation tool capable of only 60 percent automation as our example, the resultant 400,000 lines of untransformed code would require 2,500 man-days of manual intervention. If however, a transformation tool was available that provided a 99 percent level of automation, that same one million-line system would only have 10,000 lines of untransformed code to be addressed in 62.5 man-days of manual intervention – a significant qualitative and quantitative improvement. Increasing the automation level therefore, as it relates to the quality and degree of transformation completeness is highly desirable and is a significant factor in selecting the optimal automated transformation solution.

Through the application of a suite of artificial intelligence (AI) technology tools, it is now possible to achieve levels of automation often exceeding 99 percent to assess, transform, re-factor or re-engineer, and if desired, web-enable a wide variety of legacy computer programming languages, along with system databases, into modern, platform-independent object-oriented software environments (Figure 1).

Much of the work to develop such a highly automated legacy system modernization technology originated from the Air Force-funded Knowledge-Based Software Assistant (KBSA) transformation research program in the late 1980s and early 1990s. The focus of that program was research to develop highly automated processes for program specification and synthesis. Program transformation technologies were adapted to achieve a highly extensible and adaptable tool suite and technology base to support the recreation of legacy system code in a new target language with minimal manual intervention. One of the author's work in support of KBSA at

Figure 1: Legacy System Transformation Solution Space



Boeing's Research and Technology Laboratory laid the foundation for the currently available automated technology. The automation levels currently being achieved for the vast majority of transformation tasks have exceeded 99 percent.

Employing this highly automated approach, legacy systems can be modernized in a fraction of the time and cost needed by the competing alternatives discussed, thus dramatically reducing the time associated with return on investment. The application of the AI technology also reduces the technical and schedule risks associated with the modernization process, while simultaneously reducing the flow time of the project. The process provides a fully documented, functional system that is in a state to be maintained and upgraded using modern tools and software workbenches.

Examples of performance metrics for projects recently addressed using this technology include a 40-to-1 reduction in functional testing time for a JOVIAL transformation task of 250,000 lines that enjoyed an automation level of 99.98 percent and a functional test of 560,000 lines of COBOL transformed at 99.99 percent that only identified 400 *bugs* during functional testing, many of which were resident in the original COBOL system. These same automated tools have also recently been applied to the transformation of C, FORTRAN, and BAL Assembler systems with levels of automation that have been demonstrated to exceed 99.99 percent for large systems.

The newly generated software also has the benefit of consistent quality and uniformity because an automated tool created it. Systems comprised of large quantities of code, if addressed manually, will require many programmers. Programmers, though writing code in the same languages, have different styles and skills. Those differences can create major difficulties during the system integration and testing phase of a transformation project. A highly automated approach requires negligible manual intervention, offers a solution that facilitates the uniformity of the code and thus, compresses the integration and testing schedule for the project.

Technology Description

By employing AI-based automated program transformation technologies, the legacy application and database modernization process can be addressed in four disciplined steps:

- **Assessment:** Captures the legacy system's *As Is* state by extracting properties

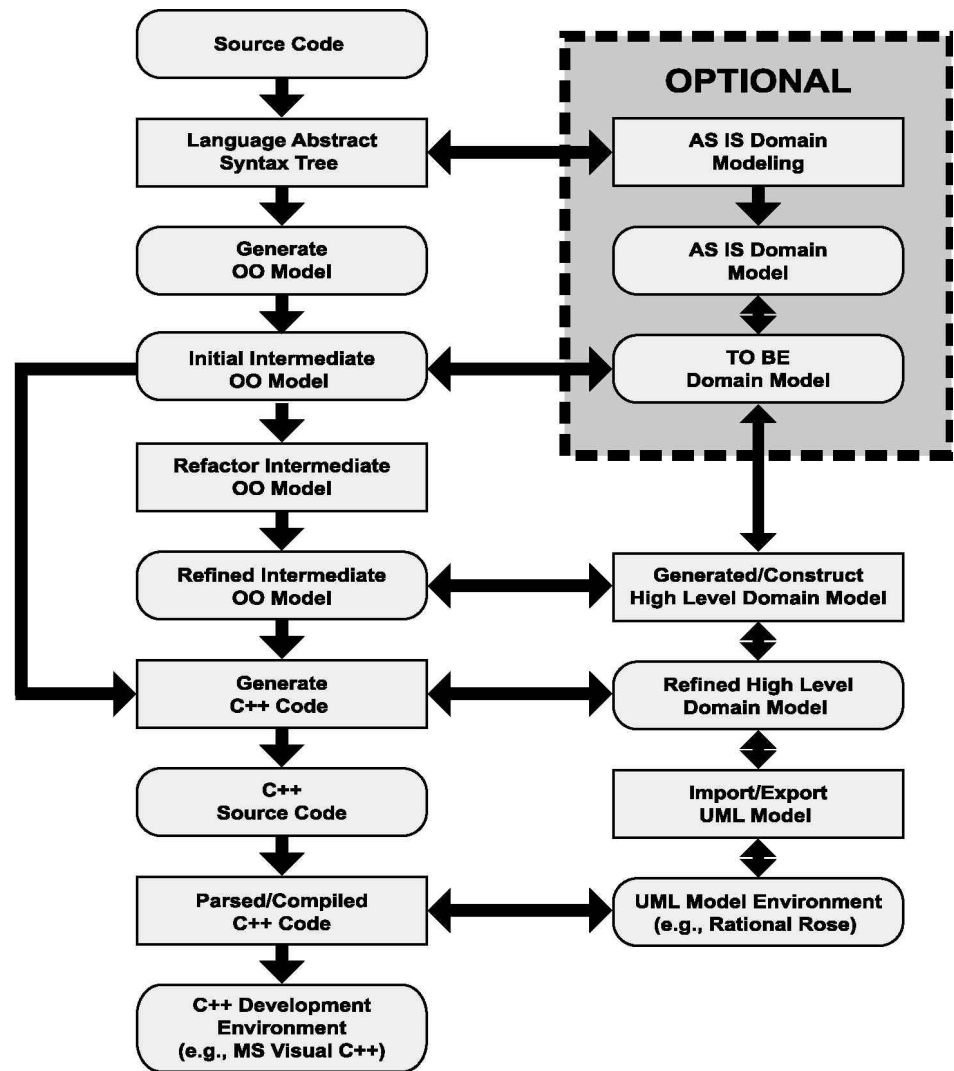


Figure 2: Automated Modernization of Legacy System into C++

of the existing system's design, and simultaneously generating detailed documentation of the system.

- **Transformation:** Provides transformed software that is compiler-ready and testable at the unit level, and fully documented.
- **Refactoring:** Reengineers the new system to improve system architecture and performance. The refactoring process provides a disciplined approach to design improvement that minimizes the chances of introducing new flaws.
- **Web-enablement:** Facilitates migration of the new system to the Web environment by transforming the legacy application to Java that runs on a Java Virtual Machine.

Figure 2 illustrates the process for automated legacy system modernization through the assessment, transformation, and refactoring processes. A discussion of these three building blocks along with associated Web-enablement of a modernized system follows.

Assessment

The code is parsed to build an in-memory knowledge-based abstract syntax tree (KBAST) model of the entire system. An inventory is developed, using an iterative process, against the KBAST model to determine if any components of the application system are missing, detect multiple versions of code, and identify linkage problems. Deviations of the dialects from standard are typically not well documented; this makes it necessary to develop a series of modifications to the parser before the technology addresses all constructs of the applications code.

After development of the KBAST model, a preliminary transformation of the source code into the target *To Be* model is performed. The purpose of this effort is to assess and compare the *As Is* and *To Be* system models to determine what modifications are to be made to the transformation process to achieve a highly automated transformation into the target language.

A *dry run* of the transformation

process is performed by creating an intermediate object-oriented (IOO) model to develop the transformation metrics, including identification of the percentage of redundant and re-usable code, current and predicted code properties, and potential code and data size reductions possible in the refactoring process. The code is then transformed into the IOO formalism that allows for detailed identification and assessment of the properties of the target system. Of key significance for reuse and maintenance is 1) extraction, parameterization, and merging of derived methods associated with derived classes; and 2) measurement of the amount of decoupling and degree of cohesion and coherence of the resultant system.

The final step of the assessment process involves domain analysis of a system, which is a process that systematically creates a common framework for describing program elements and situations within the code. This descriptive framework facilitates recognition of unique and common roles and relationships among one or more systems. The framework has two tightly related dimensions of analysis that address both the classifications of identifiers (data and structures) as well as the situations that involve their usage.

The construction of the first dimension of analysis entails describing the individual names that occur within a system as elements of common terms within a domain dictionary.

The second dimension describes the more complex relationships among elements in the form of interpretations. The interpretations are denotations for complex situations in the code. The AI-based tech-

nology automatically constructs interpretations by rewriting code directly from the structures represented by the code's abstract syntax in the KBAST. Interpretations resemble the sentential and syntactic form of the code except that domain dictionary terms have been substituted for the identifier names in the program code. A single interpretation will therefore match many syntactically similar but terminologically different specific situations. These situations may occur in the code and serve to identify commonality among complex relationships that occur within a single system or span multiple systems.

Interpretations are stored within annotation libraries and used within the technology for documenting decisions about the situations in the code. Interpretations are automatically generated for individual program statements, data structure definitions, basic code blocks, functions, or entire programs. These interpretations range in specificity from generic to domain-specific interpretations. The degree of specificity or generality in the interpretation depends upon the relative generality or specificity of the type denotation substitutions. The size of the interpretation depends upon the user-directed or system-directed choice of context of interest.

Domain analysis assists in identifying issues and opportunities in the transformation and refactoring process. It only requires manual intervention when it is necessary to establish a taxonomic system to support classification tasks. An example would be the identification or comparison of sets of code-level statement functions and data structures that are of specific

interest such as the data-related usages in an entire application.

Transformation

The modernization process begins with the automatic identification of candidate classes and objects for output into an IOO model. This is a relatively complete transformation of the input source code into an IOO model that is consistent with the structure of object-oriented (OO) C++. This transformation into the IOO form locates redundant, duplicate, and similar data and processes, and abstracts those detected items into classes and methods. The classes, relationships, attributes, and operations of the derived IOO model conform to Universal Modeling Language (UML) standards.

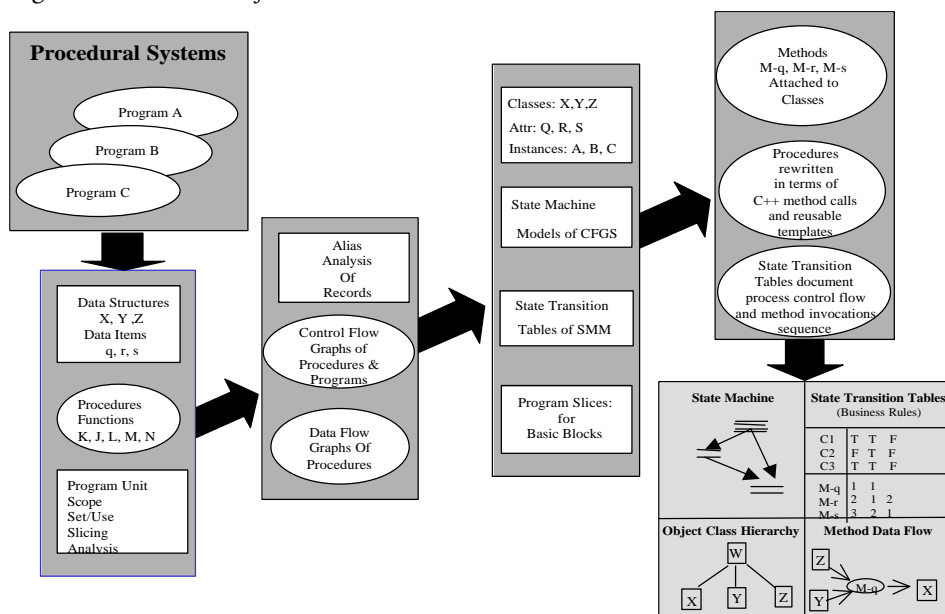
The overall process for transforming from a procedural to an OO application starts with input of legacy application programs and produces as output a completely integrated and modernized system. The output system consists of object classes and their instances that are complete with regard to data typing, methods, and IOO processes (executable mission-oriented C++ functions, which refer to class member element and member functions or methods). These constructs possess a derived architecture and control structure consistent with the OO programming paradigm. The IOO application contains calls to derived methods associated with desired classes. Figure 3 provides a more specific depiction of the IOO model generation process.

The design documentation extracted from the IOO model is a hybrid between conventional OO modeling languages and event-driven programming models. The mapping from procedural code into OO code is functionally faithful to the original procedural system. However, this IOO model follows the semantic and syntactic rules of the OO languages C++ and Java.

It should be emphasized that variations in the transformation process do not come for free. The effort of adapting the transformation technology base to customer-specific objectives typically requires tailoring the transformation pathways to customer-specific objectives. This is the principle effort performed by a transformation service provider in a highly automated system modernization process.

Support for the system transformation processes comprises a wide range of tasks. Some of the components include physical transformation of the application software, user interface, databases, and platform adaptation. Table 1 illustrates the overall

Figure 3: Architecture of the Avionics Simulation Station



set of activities in modernizing an information system. The principal roles and responsibilities of the transformation technology provider, major system integrator, and customer are indicated in the column headings.

The principal project phases are identified in the table cells in Table 1. Column one task components include application transformation, user interface transformation, database transformation, and platform adaptation, and can be addressed with a very high level of automation. The second column involves functional testing of all major components to assure functional integrity of the resultant system in the new target environment. The final column encompasses deployment and retraining of the user base.

In a typical engagement, the user interface is automatically transformed into a functionally equivalent user interface in the target environment. The database is converted using information derived from the system database descriptions that drive database conversion programs. Platform and operating system calls are transformed into equivalent target environment services.

Generally, we have found it possible to automate virtually all of the application, database, user-interface, and platform adaptation tasks. Automation has not been introduced where manual processes are already adequate such as the specification of index fields in a relational database using the interface provided by the database vendor, testing the system user-interface, or identification and reporting of system bugs and enhancements.

Throughout the development of an automated process the largest unknowns have been resolving configuration discrepancies that surfaced because the original source code was either incomplete or could not be demonstrated to build the original system, and determining the functionality of undocumented legacy system calls. The most significant areas of effort have been in the replacement of legacy platform functionality that did not exist in the target environment, and enhancing the dialect-specific language constructs.

Refactoring

Refactoring is the process of changing a software system to improve the software's structure and performance without altering its functional behavior. Refactoring is used to eliminate, replace, or rewrite code to improve its efficiency and understandability or to transform applications to use a suitable set of modern infrastructure support functions. Refactoring extracts

Transformation Technology Provider	Major Systems Integrator	Integrator and Customer
Platform Adaptation	Platform Adaptation Function Test	Platform Adaptation Deployment
User Interface Transformation	User Interface Transformation Req't's Function Test	User Interface Transformation Req't's Deployment
Data Base Transformation	Data Transformation Function Test	Data Transformation Deployment
Application Transformation	Application Functional Test	Application Deployment

Table 1: *Legacy System Modernization Services and Responsibilities*

and parameterizes methods; merges and consolidates similar methods; reduces the set of methods associated with a class to the minimal set of well-understood operations; improves the coupling, cohesion and comprehensibility of the overall application; and reduces overall code duplication and code redundancy.

Legacy applications often have many dependencies on the legacy software infrastructure. Consequently, it is often not directly portable to another software environment. Modernization of applications often requires isolating the application-specific code from the code associated with various environment-specific utility/support functions known as infrastructure code.

Often the legacy infrastructure functionality does not exist in the new environment, or exists in a different form. Thus, this layer of support services must be discontinued, redeveloped, or suitably replaced. The definition or introduction of an appropriate interface to the facilities/services layer into the newly derived application requires the development of new code or an application program interface (API) layer to comparable support services in the target platform. This is also required to test both the interface and the service layers accessed through the facilities/services layer interface.

Adaptation of the application to interface with a target environment not previously encountered is the most manually intensive effort in a transformation project. In our experience, this adaptation has been the single largest source of schedule risk, though it is not in general, a significant technical risk. Generally lessons learned from these kinds of adaptation tasks are transferable between projects. Separation of these infrastructure layers from the application layers via a suitable interface layer expedites resolution of errors that arise from variations in alternative target operational environments.

Web-Enablement

Web-enablement entails the transformation of an application into a networked,

distributed application that makes use of the following:

- Browser user-interfaces (BUI).
- Web-based languages.
- Run-time environments such as Java and the Java Virtual Machine (JVM).
- Web-based data transmission and manipulation protocols such as the Extended Markup Language for the interchange of data.

Hybrid Web applications exhibit some, but not necessarily all of these features. A Web-application may be written in C++ and have a BUI front-end that supports interface with users, or a back-end database with connectivity via Microsoft's Object Data Base Connectivity (ODBC). Java applications typically employ Java Data Base Connectivity to connect to various vendor database products with similar functionality as ODBC.

The Common Object Request Broker Architecture (CORBA) has a common interface definition language (IDL) that supports both Java and C++. CORBA is commonly used to provide distributed component services for a smaller number of users with high-performance requirements. CORBA's IDL facilitates the creation of networked distributed applications by simplifying the definition of interfaces that allow components to call one another that reside anywhere in the network, and may be implemented in any language that supports IDL. A component architecture such as J2EE provides standardized, extensible server-side and client-side components to provide multi-tier distributed services. Java more typically uses remote method invocation than CORBA IDL for transferring data between distributed components.

Support for both CORBA and Enterprise Java Beans for distributed component management services can coexist in large applications. For instance, CORBA with C++ can be used for highly optimized transaction-oriented database applications, while Java Enterprise Java Beans and Java Applets are often used for highly interactive distributed applications.

CROSSTALK

The Journal of Defense Software Engineering

Get Your Free Subscription

Fill out and send us this form.

OO-ALC/TISE
7278 FOURTH STREET
HILL AFB, UT 84056

FAX: (801) 777-8069 DSN: 777-8069

PHONE: (801) 775-5555 DSN: 775-5555

Or request online at www.stsc.hill.af.mil

NAME: _____

RANK/GRADE: _____

POSITION/TITLE: _____

ORGANIZATION: _____

ADDRESS: _____

BASE/CITY: _____

STATE: _____ ZIP: _____

PHONE: (____) _____

FAX: (____) _____

E-MAIL: _____@_____

CHECK BOX(ES) TO REQUEST BACK ISSUES:

JAN2000 ☐ LESSONS LEARNED

FEB2000 ☐ RISK MANAGEMENT

MAY2000 ☐ THE F-22

JUN2000 ☐ PSP & TSP

MAY2001 ☐ SOFTWARE ODYSSEY

JUL2001 ☐ TESTING & CM

AUG2001 ☐ SW AROUND THE WORLD

SEP2001 ☐ AVIONICS MODERNIZATION

OCT2001 ☐ OPEN & COMMON SYSTEMS

NOV2001 ☐ DISTRIBUTED SW DEV.

Given the multiple business processes performed by the applications within large confederated legacy systems and the trade-offs between several possible alternative distributed component Web-based architectures, the definition of the most appropriate transformation approach is a complex decision to be evaluated. The decision requires an in-depth analysis of the customer's applications, analysis of the implications of alternative solutions, and possibly some amount of iteration to define an appropriate transformation pathway. This decision process is driven by the current system architecture; the target architecture objectives; the technical infrastructure, including overall tool-set capabilities; and personnel resources available to support this transformation process.

Conclusion

While transforming legacy source code to C++ at an automation level of 99.99 percent is achievable using the approach described here, experience has also shown that it is unwise to take too many steps along the modernization pathway at one time. Hence, one should regard the initial transformation into C++ as an easily achievable goal that provides the staging point for subsequent phases, including system refactoring, confederated system

consolidation, and Web-enablement.

It should be emphasized that while high levels of automation are achievable for transformation tasks, a modernization project also involves development and adaptation tasks that are manually intensive such as infrastructure API layer development for unfamiliar legacy or target platforms. In addition, there are a number of tasks that by their very nature require human guidance or description such as certain forms of domain analysis and refactoring tasks that require specific domain expertise. We have however, been successful at minimizing the effort associated with these tasks by providing high levels of machine support for them as well.

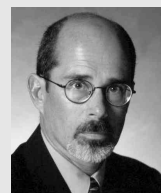
Nevertheless, as today's organizations address the critical structural, cultural, and financial issues surrounding the migration of their often irreplaceable legacy software applications and databases to modern platform-independent computing environments, it is essential that they understand that a new automated low-risk approach is available. Exceedingly advanced tool-sets and processes for rapidly reengineering legacy system software into modern computing environments provides organizations with a valuable new alternative that is faster, lower cost, lower risk, and higher quality than other methods currently available. ♦

About the Authors



Philip Newcomb is an internationally recognized expert in the application of artificial intelligence (AI) and formal methods of software engineering, and has published numerous papers in his field. He has done groundbreaking research in applying AI, software engineering, and automatic programming. He formulated the conceptual product framework and developed the software transformation technology and products offered by The Software Revolution, Inc. He has graduate work and degrees from Carnegie Mellon University, the University of Washington, Ball State University and Indiana University.

The Software Revolution, Inc.
3330 Monte Villa Pkwy.
Bothell, WA 98021
Phone: (425) 354-6464
Fax: (425) 354-6465
E-mail: newcomb@softwarerevolution.com



Randy A. Doblar, vice president, Sales and Marketing, for The Software Revolution, Inc., has more than 28 years experience in program management and business development in the defense and commercial marketplace. He has published papers in the scientific disciplines of acoustics, geophysics, and oceanography, and has been instrumental in leading The Software Revolution, Inc.'s effort to establish the eVolution 2000™ technology as a new industry standard for legacy system modernization.

The Software Revolution, Inc.
3330 Monte Villa Pkwy.
Bothell, WA 98021
Phone: (425) 354-6480
Fax: (425) 354-6465
E-mail: rdoblar@softwarerevolution.com